SYSTEM AND METHOD FOR ON-NETWORK STORAGE SERVICES

BACKGROUND OF THE INVENTION

1. Related Applications.

The present invention claims priority from U.S. Provisional Patent Application No. 60/197,490 entitled CONDUCTOR GATEWAY filed on April 17, 2000.

2. Field of the Invention.

The present invention relates, in general, to network information access and, more particularly, to software, systems and methods for providing database services in a coordinated fashion from multiple cooperating database servers.

3. Relevant Background.

Increasingly, business data processing systems, entertainment systems, and personal communications systems are implemented by computers across networks that are interconnected by internetworks (e.g., the Internet). The Internet is rapidly emerging as the preferred system for distributing and exchanging data. Data exchanges support applications including electronic commerce, broadcast and multicast messaging, videoconferencing, gaming, and the like. In electronic commerce (e-commerce) applications, it is important to provide a satisfying buying experience that leads to a purchase transaction. To provide this high level of service, a web site operator must ensure that data is delivered to the customer in the most timely, usable and efficient fashion.

The Internet is a collection of disparate computers and networks coupled together by a web of interconnections using standardized communications protocols. While most Internet access is currently performed using conventional personal computers and workstations, the variety of devices that access the Internet is growing quickly and expected to continue to grow. It is expected that a variety of appliances and devices within offices, businesses, and households will support Internet connectivity in the coming years. A major segment of growth is in the area of lightweight computing appliances. Examples include wireless telephones, personal digital assistants (PDAs), digital picture frames, digital music, and digital movies among other examples. These devices are characterized by little or no mass storage capability. In such devices there is increased need to access external mass storage such as network storage devices to access information needed to perform their functions.

The Internet is characterized by its vast reach as a result of its wide and increasing availability and easy access protocols. Unfortunately, the ubiquitous nature of the Internet results in variable bandwidth and quality of service between points. The latency and reliability of data transport is largely determined by the total amount of traffic on the Internet and so varies wildly seasonally and throughout the day. Other factors that affect quality of service include equipment outages and line degradation that force packets to be rerouted, damaged and/or dropped. Also, routing software and hardware limitations within the Internet infrastructure may create bandwidth bottlenecks even when the mechanisms are operating within specifications. The variable nature of the quality of service (QOS) provided by the Internet has made development and deployment of database system that leverage the Internet infrastructure difficult.

With the advent of the Internet, computing appliances that can potentially act as interfaces to a database have potentially ubiquitous access to this stored database information. The Internet promises to enable ready access from a wide variety of computing appliances at a wide variety of locations. Typically, when data is stored on a network it is stored at a location associated with a network service that administers that data. For example, MP3 music files may be stored in a centralized database that stores only MP3 files. Digital movies or presentation materials are stored on specific servers that administer requests for those materials. This enables the administering server to regulate, control, and charge for access to the data.

Managing access to data files often involves a disparity between the resources required to perform the administrative and management functions and the resources required to serve the data efficiently. Management functions such as receiving requests, locating files, recording metadata describing who, when and where the files were accessed, account management and billing tend to involve relatively small volumes of data that are efficiently handled by a processor with fast access to an administrative database. In contrast, the actual data file delivery involves larger data units with transactions and are beneficially performed by a processor with a low latency connection to the end-user that is receiving the data.

However, the conventional close-coupling between the services that manage the data and the data store itself restricts the accessibility of the data. This results in data stores being located behind a database management engine at a location that is not optimal for delivery of data to end users and increases the cost of transporting the data. Alternatively, management functions can be

replicated across multiple servers requiring coordination, synchronization and added complexity. A need exists for on-network data storage systems and methods that efficiently perform the disparate tasks associated with data storage and management.

Beyond varying functional requirements for data storage and access, there are increasing political, security, legislative and availability criteria that influence where certain data is physically stored or across what borders it is transported. For example, politically sensitive data may not be permitted in some jurisdictions. In another example, a law firm may wish that all client data be physically stored on servers within its control. Until now, such data storage solutions could not be managed by external services. For example, if the data owner wished to make data available for a per-access charge, the owner would be forced to implement the charging mechanisms on its own servers, or compromise the desired data storage criteria by replicating the data onto the servers of an external service provider. Hence, a need exists for systems and methods that enable an external service provider to provide data management and access services to data that is physically stored on data-owner controlled storage mechanisms.

SUMMARY OF THE INVENTION

Briefly stated, the present invention involves a method and system for managing on-network data storage using a communication network. Requests for data are received within an intermediary server from a plurality of external client applications coupled to the network. Units of data are stored in one or more data storage devices accessible to the intermediary server. Each

storage request is associated with a token representing the request. The token is sent to a storage management server coupled to the network and having an interface for communicating with the intermediary server. The storage management server returns specific location information corresponding to the request associated with the received token. The intermediary server accesses the data storage mechanism using the specific location information to retrieve data at the specific location. The retrieved data is delivered to the client application that generated the request.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a general distributed computing environment in which the present invention is implemented;

FIG. 2 shows in block-diagram form significant components of a system in accordance with the present invention;

FIG. 3 shows a network architecture of components implementing the on-network data storage system in accordance with the present invention;

FIG. 4 shows front-end components of FIG. 2 in greater detail;

FIG. 5 illustrates entity relationships and data exchanges in a first type of storage access in accordance with the present invention; and

FIG. 6 illustrates entity relationships and data exchanges in a second type of storage access in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention is illustrated and described in terms of a distributed computing environment such as an enterprise computing system using public communication channels such as the Internet. However, an important feature of the present invention is that it is readily scaled upwardly and downwardly to meet the needs of a particular application. Accordingly, unless specified to the contrary, the present invention is applicable to significantly larger, more complex network environments, including wireless network environments, as well as small network environments such as conventional LAN systems.

In accordance with the present invention, some or all of the data storage normally implemented at the web site implementing the data management processes are instead implemented in a front-end server that enjoys a lower latency connection to an end user. The administration services are handled centrally while the data storage is handled more locally to a client.

In another respect, the present invention enables separation between the tasks involved in physical storage and the tasks involved in managing access to the physical storage. From a security perspective, this enables data to be placed in a physical storage location that meets criteria defined by the data owner such as physical location, topological location, legal jurisdictions, and the like. Physical data storage may actually be implemented on the data owner's machine, or on one or more network storage device(s) that provides suitable access control or security provisions appropriate for the data. At the same time, data administration services can be implemented in a network service node independent of the physical storage. In this manner, a user data requests can be metered and managed by the network service node

while the data owner does not need to compromise the desired storage criteria.

Specific implementations of the present invention involve the use of "web server" software and hardware to implement intermediary servers. For purposes of this document, a web server is a computer running server software coupled to the World Wide Web (i.e., "the web") that delivers or serves web pages. The web server has a unique IP address and accepts connections in order to service requests by sending back responses. A web server differs from a proxy server or a gateway server in that a web server has resident a set of resources (i.e., software programs, data storage capacity, and/or hardware) that enable it to serve web pages using the resident resources whereas a proxy or gateway is an intermediary program that makes requests on behalf of a client to resources that reside elsewhere. A web server in accordance with the present invention may reference external resources of the same or different type as the services requested by a user, and reformat and augment what is provided by the external resources in its response to the user. Commercially available web server software includes Microsoft Internet Information Server (IIS), Netscape Netsite, Apache, among others. Alternatively, a web site may be implemented with custom or semi-custom software that supports HTTP traffic.

FIG. 1 shows an exemplary computing environment 100 in which the present invention may be implemented. Environment 100 includes a plurality of local networks such as Ethernet network 102, FDDI network 103 and Token Ring network 104. Essentially, a number of computing devices and groups of devices are interconnected through a network 101. For example, local networks 102, 103 and 104 are each coupled to network 101 through routers 109. LANs 102, 103 and 104 may be implemented using any available

topology and may implement one or more server technologies including, for example UNIX, Novell, or Windows NT networks, including both client-server and peer-to-peer type networks. Each network will include distributed storage implemented in each device and typically includes some mass storage device coupled to or managed by a server computer. Network 101 comprises, for example, a public network such as the Internet or another network mechanism such as a fibre channel fabric or conventional WAN technologies.

Local networks 102, 103 and 104 include one or more network appliances 107. One or more network appliances 107 may be configured as an application and/or file server. Each local network 102, 103 and 104 may include a number of shared devices (not shown) such as printers, file servers, mass storage and the like. Similarly, devices 111 may be shared through network 101 to provide application and file services, directory services, printing, storage, and the like. Routers 109 provide a physical connection between the various devices through network 101. Routers 109 may implement desired access and security protocols to manage access through network 101.

Network appliances 107 may also couple to network 101 through public switched telephone network 108 using copper or wireless connection technology. In a typical environment, an Internet service provider 106 supports a connection to network 101 as well as PSTN 108 connections to network appliances 107. The present invention may be particularly useful wireless applications because many wireless appliances 107 have limited local data storage capability which makes obtaining external data more frequent and important. The present invention enables the data to be stored nearer to the wireless appliance, for example in ISP 106, but managed by any network-connected server 111 or appliance 107.

Network appliances 107 may be implemented as any kind of network appliance having sufficient computational function to execute software needed to establish and use a connection to network 101. Network appliances 107 may comprise workstation and personal computer hardware executing commercial operating systems such as Unix variants, Microsoft Windows, Macintosh OS, and the like. At the same time, some appliances 107 comprise portable or handheld devices using wireless connections through a wireless access provider such as personal digital assistants and cell phones executing operating system software such as PalmOS, WindowsCE, EPOCOS and the like. Moreover, the present invention is readily extended to network devices such as office equipment, vehicles, and personal communicators that make occasional connection through network 101.

Each of the devices shown in FIG. 1 may include memory, mass storage, and a degree of data processing capability sufficient to manage their connection to network 101. The computer program devices in accordance with the present invention are implemented in the memory of the various devices shown in FIG. 1 and enabled by the data processing capability of the devices shown in FIG. 1. In addition to local memory and storage associated with each device, it is often desirable to provide one or more locations of shared storage such as disk farm (not shown) that provides mass storage capacity beyond what an individual device can efficiently use and manage. Selected components of the present invention may be stored in or implemented in shared mass storage.

In one embodiment, the present invention operates in a manner akin to a private network 200 implemented within the Internet infrastructure. This private network 200 is used to transport data between clients 205 and data servers 210, and/or to transport management and access

control information between storage management servers 212 and the data servers 210 and clients 205. In essence, the private network 200 enables the split of physical storage (implemented by storage server 210 and data store 211) and storage management and access control (implemented by storage management server 212) contemplated by the present invention.

Private network 200 expedites and prioritizes communications between a client 205 and a data server 210. In the exemplary implementations, two intermediary computers, front-end 201 and back-end 203 are used cooperatively as intermediary servers to process database access requests and provide data services. However, it is contemplated that a single intermediary computer (i.e., either front-end 201 or back-end 203) may be used and still provide improved access to a data server 210. Further, it is also contemplated that FE201 and data server 210 reside in the same physical location.

In the specific examples herein client 205 comprises a network-enabled graphical user interface such as a web browser. However, the present invention is readily extended to client software other than conventional web browser software. Any client application that can access a standard or proprietary user level protocol for network access is a suitable equivalent. Examples include client applications that act as front ends for file transfer protocol (FTP) services, extensible markup language (XML) services, Voice over Internet protocol (VoIP) services, network news protocol (NNTP) services, multi-purpose internet mail extensions (MIME) services, post office protocol (POP) services, simple mail transfer protocol (SMTP) services, as well as Telnet services. In addition to network protocols, the client application may serve as a front-end for a network application such as a database management system (DBMS) in which case the client

application generates query language (e.g., structured query language or "SQL") messages. In wireless appliances, a client application functions as a front-end to a wireless application protocol (WAP) service.

Data server 210 implements connectivity to network devices such as back-end 203 to receive and process requests for data from data store 211. Data server 210 can be implemented as a database including relational, flat, and object oriented databases. Alternatively, data server 210 may comprise a virtual database that accesses one or more other databases. Further, data server 210 may be a data storage device or network file system that responds to requests by fetching data.

Front-end mechanism 201 serves as an access point for client-side communications. In one example, front-end 201 comprises a computer that sits "close" to clients 205. By "close", "topologically close" and "logically close" it is meant that the average latency associated with a connection between a client 205 and a front-end 201 is less than the average latency associated with a connection between a client 205 and a server 210. Desirably, front-end computers have as fast a connection as possible to the clients 205. For example, the fastest available connection may be implemented in point of presence (POP) of an Internet service provider (ISP) 106 used by a particular client 205. However, the placement of the front-ends 201 can limit the number of browsers that can use them. Because of this, in some applications it is more practical to place one front-end computer in such a way that several POPs can connect to it. Greater distance between front-end 201 and clients 205 may be desirable in some applications as this distance will allow for selection amongst a greater number of front-ends 201 and thereby provide significantly different routes to a particular back-end 203. This may offer benefits when

particular routes and/or front-ends become congested or otherwise unavailable.

Transport mechanism 202 is implemented by cooperative actions of the front-end 201 and back-end 203. Back-end 203 processes and directs data communication to and from data server 210. Transport mechanism 202 communicates data packets using a proprietary protocol over the public Internet infrastructure in the particular example. Hence, the present invention does not require heavy infrastructure investments and automatically benefits from improvements implemented in the general-purpose network 101. Unlike the general-purpose Internet, front-end 201 and back-end 203 are programmably assigned to serve accesses to a particular data server 210 at any given time.

It is contemplated that any number of front-end and back-end mechanisms may be implemented cooperatively to support the desired level of service required by the data server owner. The present invention implements a many-to-many mapping of front-ends 201 to back-ends 203. Because the front-end to back-end mappings can by dynamically changed, a fixed hardware infrastructure can be logically reconfigured to map more or fewer front-ends to more or fewer back-ends and web sites or servers as needed.

In one embodiment, front-end 201 and back-end 203 are closely coupled to the Internet backbone. This means they have high bandwidth connections, can expect fewer hops, and have more predictable packet transit time than could be expected from a general-purpose connection. Although it is preferable to have low latency connections between front-ends 201 and back-ends 203, a particular strength of the present invention is its ability to deal with latency by enabling efficient transport and traffic prioritization. Hence, in other embodiments front-end 201

and/or back-end 203 may be located farther from the Internet backbone and closer to clients 205 and/or data servers 210. Such an implementation reduces the number of hops required to reach a front-end 201 while increasing the number of hops within the TMP link 202 thereby yielding control over more of the transport path to the management mechanisms of the present invention.

Clients 205 no longer conduct all data transactions directly with the data server 210. Instead, clients 205 conduct some and preferably a majority of transactions with front-ends 201, which access the functions of data server 210. Client data is then sent, using TMP link 202, to the back-end 203 and then to the server 210. Running multiple clients 205 over one large connection provides several advantages:

- Since all client data is mixed, each client can be assigned a priority. Higher priority clients, or clients requesting higher priority data, can be given preferential access to network resources so they receive access to the channel sooner while ensuring low-priority clients receive sufficient service to meet their needs.

- The large connection between a front-end 201 and back-end 203 can be permanently maintained, shortening the many TCP/IP connection sequences normally required for many clients connecting and disconnecting.

A particular advantage of the architecture shown in FIG. 2 is that it is readily scaled. In accordance with the present invention, not only can the data itself be distributed, but the data service functionality and behavior is readily and dynamically ported to any of a number of intermediary computers in contrast to

conventional database systems where the database functionality is confined to a particular server or limited set of servers. In this manner, any number of client machines 205 may be supported. In a similar manner, a database owner may choose to use multiple data servers 210 that are co-located or distributed throughout network 101. To avoid congestion, additional front-ends 201 may be implemented or assigned to particular data servers. Each front-end 201 is dynamically re-configurable by updating address parameters to serve particular data servers. Client traffic is dynamically directed to available front-ends 201 to provide load balancing. Hence, when quality of service drops because of a large number of client accesses to a particular data server, an additional front-end 201 can be assigned to the data server and subsequent client requests directed to the newly assigned front-end 201 to distribute traffic across a broader base.

In the particular examples, this is implemented by a front-end manager component 207 that communicates with multiple front-ends 201 to provide administrative and configuration information to front-ends 201. Each front-end 201 includes data structures for storing the configuration information, including information identifying the IP addresses of data servers 210 to which they are currently assigned. Other administrative and configuration information stored in front-end 201 may include information for prioritizing data from and to particular clients, quality of service information, and the like.

Similarly, additional back-ends 203 can be assigned to a data server to handle increased traffic. Back-end manager component 209 couples to one or more back-ends 203 to provide centralized administration and configuration service. Back-ends 203 include data structures to hold

current configuration state, quality of service information and the like. In the particular examples front-end manager 207 and back-end manager 209 serve multiple data server 210 and so are able to manipulate the number of front-ends and back-ends assigned to each data server 210 by updating this configuration information. When the congestion for the data server subsides, the front-end 201 and back-end 203 can be reassigned to other, busier data servers. These and similar modifications are equivalent to the specific examples illustrated herein.

In the case of web-based environments, front-end 201 is implemented using custom or off-the-shelf web server software. Front-end 201 is readily extended to support other, non-web-based protocols, however, and may support multiple protocols for varieties of client traffic. Front-end 201 processes the data traffic it receives, regardless of the protocol of that traffic, to a form suitable for transport by TMP 202 to a back-end 203. Hence, most of the functionality implemented by front-end 201 is independent of the protocol or format of the data received from a client 205. Hence, although the discussion of the exemplary embodiments herein relates primarily to front-end 201 implemented as a web server, it should be noted that, unless specified to the contrary, web-based traffic management and protocols are merely examples and not a limitation of the present invention.

As shown in FIG. 2, in accordance with the present invention data access services are implemented using an originating data server 210 operating cooperatively with the server of front-end 201 and a storage management server 212. Some or all of the functions of storage management server 212 may be implemented in a front-end 201, although such implementation diverges from the centralized storage management provided by storage management server 212.

Front-ends 201 alone or in cooperation with one or more back-ends 203 function as intermediary servers 206 as shown in Fig. 3. The abstraction of Fig. 3 simplifies the complexity of the implementation of private network 200. This abstraction is useful because the intermediary servers 206 integrate the otherwise separated physical storage and storage management components such that a client 205 can make data requests and receive data responses without requiring knowledge of the private network 200 shown in Fig. 2. In the preferred implementations, the physical data store 211 is coupled to an intermediary server 206 by a low latency connection. This low latency connection may be a connection through private network 200 as shown in Fig. 2, as well as LAN, WAN, MAN or SAN connections. It is also contemplated that physical data store 211 may be directly connected to intermediary server 206. Regardless of the location of physical data store 211, intermediary server 206 references storage management server 212 in order to locate particular files needed to respond to requests.

In order for a client 205 to obtain service from an intermediary server 206, it must first be directed to an intermediary server 206 (e.g., a front-end server 201) that can provide the desired service. Preferably, client 205 does not need to be aware of the location of intermediary server 206, and initiates all transactions as if it were contacting the storage server 210. In a particular implementation, a domain name server (DNS) redirection mechanism is used to connect a client 205 to a particular intermediary server 206. The DNS systems is defined in a variety of Internet Engineering Task Force (IETF) documents such as RFC0883, RFC 1034 and RFC 1035 which are incorporated by reference herein. In this implementation, at least one DNS server 307 is owned and controlled by system components of the present invention.

When a user accesses a network resource (e.g., a makes a data request), client 205 contacts the public DNS system to resolve the requested domain name into its related IP address in a conventional manner. In a first embodiment, the public DNS performs a conventional DNS resolution directing the browser to an originating server 210 and server 210 performs a redirection of the browser to the system owned DNS server (i.e., DNC_C in FIG. 3). In a second embodiment, domain:address mappings within the DNS system are modified such that resolution of the originating server's domain automatically return the address of the system-owned DNS server (DNS_C). Once a browser is redirected to the system-owned DNS server, it begins a process of further redirecting the browser 301 to a selected intermediary server 206. The intermediary server 206 may be selected based on contents of its local storage, or other criteria.

Primary functions of the intermediary server 206 include responding to data requests from clients 205 by identifying the location of the stored data, accessing the stored data (i.e., performing read and/or write operations), and communicating results of the data access to the requesting client 205. Optionally, intermediary server 206 may prioritize amongst multiple queries, and resolving the queries in an order based upon the prioritization. It is contemplated that the various functions described in reference to the specific examples may be implemented using a variety of data structures and programs operating at any location in a distributed network. For example, a front-end 201 or intermediary server 206 may be operated on a network appliance 107 or server within a particular network 102, 103, or 104 shown in FIG. 1.

Fig. 4 specifically illustrates components of a front-end 201, however, it should be understood that the

components are largely similar to an intermediary server 206 with the variations noted herein. Back-end 203 provides complementary services and functions to front-end 201, and is not illustrated separately herein.

TCP component 401 includes devices for implementing physical connection layer and Internet protocol (IP) layer functionality. Current IP standards are described in IETF documents RFC0791, RFC0950, RFC0919, RFC0922, RFC792, RFC1112 that are incorporated by reference herein. For ease of description and understanding, these mechanisms are not described in great detail herein. Where protocols other than TCP/IP are used to couple to a client 205, TCP component 401 is replaced or augmented with an appropriate network protocol process.

TCP component 401 communicates TCP packets with one or more clients 205. Received packets are coupled to parser 402 where the Internet protocol (or equivalent) information is extracted. TCP is described in IETF RFC0793 which is incorporated herein by reference. Each TCP packet includes header information that indicates addressing and control variables, and a payload portion that holds the user-level data being transported by the TCP packet. The user-level data in the payload portion typically comprises a user-level network protocol datagram.

Parser 402 analyzes the payload portion of the TCP packet. In the examples herein, HTTP is employed as the user-level protocol because of its widespread use and the advantage that currently available browser software is able to readily use the HTTP protocol. In this case, parser 402 comprises an HTTP parser. More generally, parser 402 can be implemented as any parser-type logic implemented in hardware or software for interpreting the contents of the payload portion. Parser 402 may implement

file transfer protocol (FTP), mail protocols such as simple mail transport protocol (SMTP) and the like. Any user-level protocol, including proprietary protocols, may be implemented within the present invention using appropriate modification of parser 402.

To improve performance, front-end 201 optionally includes a caching mechanism 403. Cache 403 may be implemented as a passive cache that stores frequently and/or recently accessed database content or as an active cache that stores database content that is anticipated to be accessed. Upon receipt of a TCP packet, HTTP parser 402 determines if the packet is making a request for data within cache 403. If the request can be satisfied from cache 403 the data is supplied directly without reference to data server 210 (i.e., a cache hit). Cache 403 implements any of a range of management functions for maintaining fresh content. For example, cache 403 may invalidate portions of the cached content after an expiration period specified with the cached data or by data sever 210. Also, cache 403 may proactively update the cache contents even before a request is received for particularly important or frequently used data from data server 210. Cache 403 evicts information using any desired algorithm such as least recently used, least frequently used, first in/first out, or random eviction. When the requested data is not within cache 403, a request is processed to data server 210, and the returned data may be stored in cache 403.

The formulated query is passed to storage server interface 405 which handles communication with storage server 210 and storage management server 212. Channel 202 is compatible with an interface to data server 210 which may include a TCP/IP interface as well as Ethernet, Fibre channel, or other available public or proprietary physical and transport layer interfaces.

Storage server 210 and storage management server 212 return responses to interface 405 which are then supplied to data filter 406 and/or HTTP reassemble component 407. Data filter component 406 may filter and/or constrain database contents returned in the response. Data filter component 406 is optionally used to implement data decompression where appropriate, decryption, and handle caching when the returning data is of a cacheable type. HTTP reassemble component 407 formats the response into a format suitable for use by client 205, which in the particular examples herein comprises a web page transported via HTTP.

Where a front-end 201 and back-end 203 together are used to implemented an intermediary server 206, front-end 201 is responsible for translating transmission control protocol (TCP) packets from client 205 into transmission morphing protocol (TMP) packets used in the system in accordance with the present invention. Transport morphing protocol and TMP are trademarks or registered trademarks of Circadence Corporation in the United States and other countries. TMP packets comprise multiple blended requests generated by data blender 404. Blender 404 slices and/or coalesces the data portions of the received packets into a more desirable "TMP units" that are sized for transport through the TMP mechanism 202. The data portion of TCP packets may range in size depending on client 205 and any intervening links coupling client 205 to TCP component 401. Moreover, where compression is applied the compressed data will vary in size depending on the compressibility of the data. Data blender 404 receives information from front-end manager 207 that enables selection of a preferable TMP packet size. Alternatively, a fixed TMP packet size can be set that yields desirable performance across TMP mechanism 202. Data blender 404

also marks the TMP units so that they can be re-assembled at the receiving end.

Data blender 404 also serves as a buffer for storing packets from all clients 205 that are associated with front-end 201. Blender 404 mixes data packets coming into front-end 201 into a cohesive stream of TMP packets sent to back-end 203 over TMP link 202. In creating a TMP packet, blender 404 is able to pick and choose amongst the available client packets so as to prioritize some client packets over others. Prioritization is effected by selectively transmitting request and response data from multiple sources in an order determined by an priority value associated with the particular request and response. For purposes of the present invention, any algorithm or criteria may be used to assign a priority.

Also, where a front-end 201 and back-end 203 together are used to implement an intermediary server 206, storage server interface 405 can implement transport protocol algorithms that create a more efficient connection between a front-end 201 and a back-end 203. Where a single intermediary server 206 is used, however, interface 405 should implement protocols that enable communication with storage servers 210 and storage management servers 212.

Optionally, front-end 201, back-end 203, and/or intermediary computer 206 implement security processes, compression processes, encryption processes and the like to condition the received data for improved transport performance and/or provide additional functionality. These processes may be implemented within any of the functional components (e.g., data blender 404) or implemented as separate functional components within front-end 201. Also, parser 402 may identify priority information transmitted with a request. The prioritization value may be provided by the owners of data

server 210, for example, and may be dynamically altered, statically set, or updated from time to time to meet the needs of a particular application. Moreover, priority values may be computed to indicate aggregate priority over time, and/or combine priority values from different sources to compute an effective priority for each database request.

TMP is a TCP-like protocol adapted to improve performance for multiple channels operating over a single connection. The TMP mechanism in accordance with the present invention creates and maintains a stable connection between two processes for high-speed, reliable, adaptable communication. Another feature of TMP is its ability to channel numerous TCP connections through a single TMP connection 202. The environment in which TMP resides allows multiple TCP connections to occur at one end of the system. These TCP connections are then combined into a single TMP connection. The TMP connection is then broken down at the other end of the TMP pipe 202 in order to traffic the TCP connections to their appropriate destinations. TMP includes mechanisms to ensure that each TMP connection gets enough of the available bandwidth to accommodate the multiple TCP connections that it is carrying.

FIG. 5 illustrates data exchanges between entities in an exemplary data access transaction in accordance with the present invention. From the perspective of client 205, there is a simple request/response exchange that is conducted with a front-end 201 (or intermediary server 206). It is a valuable feature that client 205 need only be configured to communicate conventional request/response exchanges as this usually enables a client 205 to use the present invention without modification from existing network interface mechanisms. In other words, the client

does not need to implement specialized hardware or software.

In a web-based environment, client 205 displays a web page having a number of hypertext links. The web page is generated by any of a web server, front-end 201, or stored internally to the client. In a particular example, the links include references to desired data objects. Because these links do not refer directly to a server/directory/file name at which the data object is located, they are referred to herein as "tokens". Client 205 need not be aware of the actual location at which a data object is stored. The term "data object" as used herein refers broadly to any set of data stored at one or more specific locations within network-connected or direct connected storage mechanisms. Data objects include single files, portions of files, sequences of files, and the like.

Front-end 201 receives client data requests and implements processes to resolve the data request into a response. In cases of write operations, the response may be a confirmation or acknowledgment that the data was written to storage. In the case of a read request, the response will include requested data. However, front-end 201 lacks a priori knowledge of where the requested data resides, even if the requested data resides on direct connected storage such as data store 211 or on a virtual database 211.

Front-end 201 sends or forwards the token associated with the data request to storage management server 212. The token comprises a data structure that identifies the data that is subject of the request, and optionally identifies the requester (i.e., the client 205) and other data required by storage management server 212. In other embodiments, the token identifies on or more intended

recipients of the data, which may include the requesting client 205. In response, storage management server 212 sends file location information to front-end 201. Front-end 201 can use the file location information to locate and access the physical storage device upon which the data is stored. For example, front-end 201 generates file requests and receives file responses from a particular data store 211. Front-end 201 then sends generates and sends a response to the requesting client 205.

FIG. 6 illustrates data exchanges between entities in an exemplary data access transaction in accordance with an alternative implementation of present invention. In the implementation of Fig. 6, the front-end 201 to which the original client data request is directed does not have access to the requested data. However, front-end 201 is not aware of this until it accesses storage management sever 212 as described above. In the implementation of Fig. 6, front-end 201 uses the file location information to generate a redirect response to client 205 that points the client 205 to an alternative front-end 201 that can access the requested data. Protocols such as HTTP include redirect mechanisms that make the operation shown in Fig. 6 practical to implement without changes to the software on client 205.

In response to the redirect from the first front-end 201, client 205 generates a redirected request to the alternate front-end 201. The redirected request may include additional file location information that would allow the alternative front-end 201 to access its available data store(s) 211 directly. Alternatively, the alternate front-end 201 may refer to storage management server 212 to obtain file location information in a manner similar to that described for Fig. 5. In either case, the alternate front-end 201 supplies the response to the client's data request.

The present invention supports a variety of implementations that meet specific needs of specific applications. In the primary embodiments described above, data is served to client in response to a client request. Alternatively, data may be served to a computer other than the client, such as participants in an online meeting, broadcast, or multicast session either in response to client requests, or according to a programmed routine executing in a front-end 201. Hence, the present invention readily supports transfer of data objects from a network-connected data storage mechanism to any specified network-connected computer rather than simply returning data to the computer that requested the data object. This can be useful in presentations and multimedia distribution using broadcast and multicast. For example, a first client 205 may issue a token that represents particular data object to a front-end 201. The token may be accompanied by an identification of one or more recipients for the data object. Alternatively, the front-end 201 may maintain the identification of recipients.

As yet another alternative, the client request including a particular token may serve as trigger for further data transfers between and among front-ends 201, data stores 211, and clients 205. Certain types of data objects are either explicitly or implicitly related to other data objects. A presentation or online event, for example, often has an explicit flow such that once a particular event is reached, the system can know with a high level of certainty what possible data object or objects will be requested after the current event. By way of example of an implicit data ordering, an initial client request may include a token identifying a particular multimedia file such as a song or music video. It can be anticipated that other songs on the same album

or songs of a similar genre or artist are likely to be subject of subsequent requests.

In accordance with an embodiment of the present invention, the initial token sent by a client can be resolved to a pointer not only to the particular data object that is subject of the current request, but can include a secondary token indicating other data objects with a probability of being requested in the future. This feature enables front-end 201 to proactively redistribute data into its cache 403, for example, so that if and when the subsequent request is received, it can be served more quickly. Alternatively, the front-end 201 and/or storage management server may determine which data objects should be proactively distributed to front-end 201 in which case client 205 need not send secondary tokens.

Although the invention has been described and illustrated with a certain degree of particularity, it is understood that the present disclosure has been made only by way of example, and that numerous changes in the combination and arrangement of parts can be resorted to by those skilled in the art without departing from the spirit and scope of the invention, as hereinafter claimed. For example, while devices supporting HTTP data traffic are used in the examples, the HTTP devices may be replaced or augmented to support other public and proprietary protocols including FTP, NNTP, SMTP, SQL and the like. In such implementations the front-end 201 and/or back end 203 are modified to implement the desired protocol. Moreover, front-end 201 and back-end 203 may support different protocols such that the front-end 201 supports, for example, HTTP traffic with a client and the back-end supports a DBMS protocol such as SQL. Such implementations not only provide the advantages of the present invention, but also enable a client to access a

rich set of network resources with minimal client software.